**1. What is the purpose of RecordReader in Hadoop?**

**Ans :** While processing data in Hadoop we use Mapper and Reducer (classic MapReduce).

Both these mapper and Reducer function have <key,Value > pairs as input and output.

While storing data in HDFS its just like a mere dump. Hadoop doesn’t care of where the line ends or record ends.. it just divides the data into blocks and save.

So in order Mapper to launch , the raw data should be read . While talking about inputformat , lets consider Text for now, The record reader comes into picture.

Being launched on an input split, the record reader looks for the start of the line in the split and reads data through the split. If it is unable to find the end of the line in that input split it will read the remaining part of the line remotely from another split.

It generates meaningful data understandable by mappers and pass it to mapper.

Same is the case when inputformat is DB , it reads records in place of lines.

For example, while doing word count program my file has data like

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*I am learning HAdoop. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

And while dividing into blocks my data got splitted at learning. like

“\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*I am learning” in one block and “HAdoop. \*\*\*\*\*\*\*\*\*\*\*\*\*”

So REcord reader forms one complete line “\*\*\* I am learning HAdoop. ” and submits to the mapper.

**2. What happens if the number of reducers is 0?**

**Ans :** In this case the outputs of the map-tasks go directly to the FileSystem, into the output path set by setOutputPath(Path). The framework does not sort the map-outputs before writing them out to the FileSystem.

**3. What is meant by Map-side and Reduce-side join in Hadoop?**

**ANS :** Map-Side Joins:-

The inputs for to each map must be partitioned and sorted in a specific way. Each input dataset must be divided into the same number of partitions, and it must be sorted by the same key (the join key) in each source.

All the records for a particular key must reside in the same partition and which is mandatory. A map-side join can be used to join the outputs of several jobs that had the same number of reducers, the same keys and output files that are no bigger than the HDFS block size.

Using the org.apache.hadoop.mapred.join.CompositeInputFormat class we can achieve this. The join type (Inner or Outer) is configurable using the join expression.

For example:

func ::= tbl(<class>,”<path>”);

We can set it to the CompositeInputFormat using,

inner(tbl(org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat.class,

“hdfs://localhost:8000/usr/data”),

tbl(org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat.class,

“hdfs://localhost:8000/usr/activity”));

We can achieve following kind of joins using Map-Side techniques,

1) Inner Join

2) Outer Join

3) Override – MultiFilter for a given key, prefered values from the right source

Reduce-Side Joins:-

Reduce-Side joins are more simple than Map-Side joins since the input datasets need not to be structured. But it is less efficient as both datasets have

to go through the MapReduce shuffle phase. the records with the same key are brought together in the reducer. We can also use the Secondary Sort technique

to control the order of the records.

**4. What is the significance of conf.setMapper class?**

**5. Give an example scenario on the usage of counters.**

**Ans :** Hadoop MapReduce Counter provides a way to measure the progress or the number of operations that occur within MapReduce programs. Basically, MapReduce framework provides a number of built-in counters to measure basic I/O operations, such as FILE\_BYTES\_READ/WRITTEN and Map/Combine/Reduce input/output records. These counters are very useful especially when you evaluate some MapReduce programs. Besides, the MapReduce Counter allows users to employ your own counters. Since MapReduce Counters are automatically aggregated over Map and Reduce phases, it is one of the easiest way to investigate internal behaviors of MapReduce programs. In this post, I’m going to introduce how to use your own MapReduce Counter. The example sources described in this post are based on Hadoop 0.21 API.

Incrementing your counter

For your own MapReduce counter, you first define a enum type as follow:

public static enum MATCH\_COUNTER {

INCOMING\_GRAPHS,

PRUNING\_BY\_NCV,

PRUNING\_BY\_COUNT,

PRUNING\_BY\_ISO,

ISOMORPHIC

};

And then, when you want to increment your own counter, you should call the increment method as follows:

context.getCounter(MATCH\_COUNTER.INCOMING\_GRAPHS).increment(1);

You can access context instance within setup, cleanup, map, and reduce method in Mapper or Reducer class. You can get a desired counter via calling context.getCounter method with some enum value.

**6. Elaborate some problems which can only be solved by MapReduce and cannot be solved by PIG?**

**ANS :** Let us take a scenario where we want to count the population in two cities. I have a data set and sensor list of different cities. I want  to count the population by using one mapreduce for two cities. Let us assume that one is Bangalore and the other is Noida. So I need to consider key of  Bangalore city  similar to Noida through which I can bring the population data of these two cities to one reducer. The idea behind this is some how I have to instruct map reducer program – whenever you find city with the name ‘*Bangalore*‘ and city with the name ‘*Noida’*,  you create the alias name which will be the common name for these two cities so that  you create a common key for both the cities and it get passed to the same reducer. For this, we have to write  *custom partitioner*.

In mapreduce when you create a ‘*key’* for city,  you have to consider*‘city’* as the key. So, whenever the framework comes across a different city, it considers it as a different key. Hence, we need to use customized partitioner. There is a provision in mapreduce only, where you can write your custom partitioner and mention if city = bangalore or noida then pass similar hashcode.  However, we cannot create custom partitioner in Pig. As Pig is not a framework, we cannot direct execution engine to customize the partitioner. In such scenarios, MapReduce works better than Pig.

**7. In what kind of scenarios, MR jobs will be more useful than PIG?**

**ANS :** Answer: Let us take a scenario where we want to count the population in two cities. I have a data set and sensor list of different cities. I want to count the population by using one mapreduce for two cities. Let us assume that one is Bangalore and the other is Noida. So I need to consider key of Bangalore city similar to Noida through which I can bring the population data of these two cities to one reducer. The idea behind this is some how I have to instruct map reducer program – whenever you find city with the name ‘Bangalore‘ and city with the name ‘Noida’, you create the alias name which will be the common name for these two cities so that you create a common key for both the cities and it get passed to the same reducer. For this, we have to write custom partitioner.

In mapreduce when you create a ‘key’ for city, you have to consider ’city’ as the key. So, whenever the framework comes across a different city, it considers it as a different key. Hence, we need to use customized partitioner. There is a provision in mapreduce only, where you can write your custom partitioner and mention if city = bangalore or noida then pass similar hashcode. However, we cannot create custom partitioner in Pig. As Pig is not a framework, we cannot direct execution engine to customize the partitioner. In such scenarios, MapReduce works better than Pig.

**8. What are combiners and when are these used in a MapReduce job?**

**Ans :** The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

Here is a brief summary on how MapReduce Combiner works −

A combiner does not have a predefined interface and it must implement the Reducer interface’s reduce() method.

A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.

A combiner can produce summary information from a large dataset because it replaces the original Map output.

Although, Combiner is optional yet it helps segregating data into multiple groups for Reduce phase, which makes it easier to process.